Legal Notice

The Zoom session for this class will be recorded and made available asynchronously on Canvas to registered students.
Announcements

1. HW 4 is due Wednesday!
2. HW 5 is available! It uses Sage! You should try to install Sage ASAP to make sure you can run the code. Recently Sage has become more broken, particularly on OS X, so you might need to use a Linux VM instead.
3. HW 5 is considered *hard*, particularly if the number theory is new to you. So start early!
Last time:

• Modular exponentiation: Given $a$ compute $g^a \mod p$. Efficient to compute.
• Discrete log: Given $g^a \mod p$ compute $a$. Not efficient to compute.

This time:

• Diffie-Hellman key exchange
• Elementary discrete log algorithms
“We stand today on the brink of a revolution in cryptography.”

— Diffie and Hellman, 1976
New Directions in Cryptography

Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER, IEEE
The symmetric cryptography we just covered

AuthenticatedEncryption\(_k(m)\)
Public key crypto idea #1: Key exchange

Solving key distribution without trusted third parties

1. Alice, Bob exchange messages
2. A, B derive shared secret from messages

\[ k = \text{KDF}(\text{Kex}) \]

\[ k = \text{KDF}(\text{Kex}) \]

3. Passive eavesdropper can't recover shared secret from exchanged messages
Textbook Diffie-Hellman

[Diffie Hellman 1976]

Desired group properties:
1. Efficient "exponentiation" in group
2. "Discrete log" is hard
3. Commutativity in "exponent"

Public Parameters

$G$ a cyclic group (e.g. a subgroup of $\mathbb{F}_p^*$, or an elliptic curve)
$g$ group generator

Key Exchange

Coming up: Supersingular isogeny Diffie-Hellman

totally different structure

$g^a \in G$

$g^b$

$g^{ab}$

$g^{ab} = (g^a)^b$

Shared secret
Prime Field Diffie-Hellman

Public Parameters

\( p \) a prime
\( q \) a subgroup order; \( q \mid (p - 1) \)
\( g \in \mathbb{F}_p^* \) a generator of subgroup of order \( q \)

Key Exchange

\( g^a \mod p \)
\( g^b \mod p \)

\( g^{ab} \mod p \) a shared secret

\( a \in \mathbb{Z} \)
\( b \in \mathbb{Z} \)
The discrete log assumption

\[ G \text{ a cyclic group with generator } g. \]

\[ A \xrightarrow{g^a} C \]

Discrete log assumption:
Pr[A wins] is negligible.
The discrete log assumption

\[ g^a \rightarrow A \quad g^{a^i} \rightarrow C \]

Discrete log assumption:
\[ \Pr[A \text{ wins}] \text{ is negligible.} \]

- Discrete log is easy \( \implies \) Diffie-Hellman is easy to break.
  (Compute \( a \) and then compute public value.)

- But! Diffie-Hellman easy to break \( \nRightarrow \) discrete log is easy.

- Discrete log is in NP and coNP \( \implies \) not NP-complete

- Shor’s algorithm solves discrete log with a quantum computer in polynomial time.
Computational Diffie-Hellman Assumption

G a cyclic group with generator g.

CDH assumption:
Pr[A wins] is negligible.

- Equivalent to computing the shared Diffie-Hellman secret.
- Discrete log is easy $\implies$ CDH is false.
- Not known if equivalent to computing discrete log
- It is hard to even verify a correct solution. $(g^a, g^b, g^{ab})$
Decisional Diffie-Hellman Assumption

$G$ a cyclic group with generator $g$.

\[ A \xleftarrow{\leftarrow} g^a, g^b, w_z \]
\[ z' \]
\[ C \]
\[ z \in \{0, 1\} \]
\[ w_0 = g^{ab} \]
\[ w_1 = g^c \]

DDH assumption:
\[ |\Pr[A = 1 \mid z = 0] - |\Pr[A = 1 \mid z = 1]| \text{ negligible.} \]

- DDH hard $\implies$ difficult to distinguish Diffie-Hellman triples from random group elements.
- CDH $\implies$ DDH $\implies$ DDH hard $\implies$ CDH hard
- Assumption used for most security proofs.
Secure group choices for Diffie-Hellman

Why do we think discrete log is hard? Because there are groups for which there are no “good” algorithms known to compute them.

Group parameters are designed to avoid known cryptanalytic attacks.

Discrete log remains the best way to break Diffie-Hellman and related cryptosystems.

Types of groups used for Diffie-Hellman in practice now:

- “Prime-field Diffie-Hellman”: Subgroups of \( \mathbb{Z}_p^* \).
- “Elliptic curve Diffie-Hellman”: Elliptic curve groups
Discrete log algorithms

Three families of discrete log algorithms:

1. Algorithms whose running time depends on the size of the order of the subgroup.
   - Good for computing discrete logs in subgroups of small or smooth order. "smooth" = many small prime factors
   - Includes generic group algorithms: Baby step giant step, Pollard rho.
Discrete log algorithms

Three families of discrete log algorithms:

1. Algorithms whose running time depends on the size of the order of the subgroup.
   - Good for computing discrete logs in subgroups of small or smooth order.
   - Includes generic group algorithms: Baby step giant step, Pollard rho.

2. Algorithms whose running time depends on the size of the log.
   - Good for computing discrete logs in a known small interval.
   - Pollard lambda algorithm.
Discrete log algorithms

Three families of discrete log algorithms:

1. Algorithms whose running time depends on the size of the order of the subgroup.
   - Good for computing discrete logs in subgroups of small or smooth order.
   - Includes generic group algorithms: Baby step giant step, Pollard rho.

2. Algorithms whose running time depends on the size of the log.
   - Good for computing discrete logs in a known small interval.
   - Pollard lambda algorithm.

3. Algorithms whose running time depends on the size of the modulus. For prime field \((\mathbb{Z}_p^*)\).
   - Good for computing discrete logs in subgroups of large order.
   - Number field sieve algorithm. (index calculus)
Computing Discrete Logs in $O(\sqrt{q})$ time

**Input:** Target $t$, prime $p$, group gen $g$, order $q$

**Goal:** Solve $g^\ell \equiv t \mod p$. **Find $\ell$.**

Baby-Step Giant-Step Algorithm

1. “Giant steps”: Compute $g^0, g^{\sqrt{q}}, g^{2\sqrt{q}}, \ldots g^{\sqrt{q}^2} \mod p$.

```
  g^0[\sqrt{q}]  g^1[\sqrt{q}]  g^2[\sqrt{q}]  g^3[\sqrt{q}]  g^4[\sqrt{q}]  \ldots
```

2. “Baby steps”: Compute $tg^1, tg^2, \ldots$ until collision.

3. Solve for $\log t$ from collision: $tg^j = g^i \mod q$ find $l$. 
Computing Discrete Logs in $O(\sqrt{q})$ time

**Input:** Target $t$, prime $p$, group gen $g$, order $q$

**Goal:** Solve $g^\ell \equiv t \mod p$.

**Baby-Step Giant-Step Algorithm**

1. “Giant steps”: Compute $g^0, g^{\lfloor \sqrt{q} \rfloor}, g^{2\lfloor \sqrt{q} \rfloor}, \ldots, g^{\lfloor \sqrt{q} \rfloor^2} \mod p$.

   $$
   g^0, g^{\lfloor \sqrt{q} \rfloor}, g^{2\lfloor \sqrt{q} \rfloor}, g^{3\lfloor \sqrt{q} \rfloor}, g^{4\lfloor \sqrt{q} \rfloor},
   $$

2. “Baby steps”: Compute $tg^1, tg^2, \ldots \mod p$ until collision.

   $$
   g^0, g^{\lfloor \sqrt{q} \rfloor}, g^{2\lfloor \sqrt{q} \rfloor}, g^{3\lfloor \sqrt{q} \rfloor}, g^{4\lfloor \sqrt{q} \rfloor},
   $$

   $$
   tg^1, tg^2, \ldots, tg^5
   $$
Computing Discrete Logs in $O(\sqrt{q})$ time

**Input:** Target $t$, prime $p$, group gen $g$, order $q$

**Goal:** Solve $g^\ell \equiv t \mod p$.

**Baby-Step Giant-Step Algorithm**

1. “Giant steps”: Compute $g^0, g^{\lfloor \sqrt{q} \rfloor}, g^{2\lfloor \sqrt{q} \rfloor}, \ldots g^{\lfloor \sqrt{q} \rfloor^2} \mod p$.

2. “Baby steps”: Compute $tg^1, tg^2, \ldots \mod p$ until collision.

3. Solve for $\log t$ from collision:

   $$ tg^j = g^{i\lfloor \sqrt{q} \rfloor} \mod p $$

   $$ t = g^{i\lfloor \sqrt{q} \rfloor - j} \mod p $$

   $$ \log t = i\lfloor \sqrt{q} \rfloor - j \mod q $$
Baby Step Giant Step running time

- Storage: $O(\sqrt{q})$ group elements
- Running time: $O(\sqrt{q})$ group multiplications
- Running time: $O(\sqrt{q})$ table lookups

Total: $O(\sqrt{q} \text{ polylog } p)$

Exponential because $\sqrt{q} = (2^{\log q})^{1/2}$

Size of elements is $\log p$. 
Baby Step Giant Step running time

- Storage: $O(\sqrt{q})$ group elements
- Running time: $O(\sqrt{q})$ group multiplications
- Running time: $O(\sqrt{q})$ table lookups

Total: $O(\sqrt{q} \text{ polylog } q)$

Idea: Reduce storage using random walk.
Pollard rho for discrete log

**Input:** Target $t$, prime $p$, group gen $g$, order $q$

**Goal:** Solve $g^\ell \equiv t \mod p$. Find $\ell$.

- Take a random walk on elements $t^{a_i}g^{b_i}$.

- A collision solves the problem:

\[
t^{a_i}g^{b_i} = t^{a_j}g^{b_j} \mod p
\]

\[
t= s^\ell \rightarrow a_i \ell + b_i \equiv a_j \ell + b_j \mod q
\]

\[
\ell \equiv (b_j - b_i)(a_i - a_j)^{-1} \mod q
\]
Pollard rho for discrete log

**Input:** Target $t$, prime $p$, group gen $g$, order $q$

**Goal:** Solve $g^\ell \equiv t \mod p$.

- Take a random walk on elements $t^{a_i}g^{b_i}$.

- A collision solves the problem:

  $$ t^{a_i}g^{b_i} = t^{a_j}g^{b_j} \mod p $$

  $$ a_i\ell + b_i \equiv a_j\ell + b_j \mod q $$

  $$ \ell \equiv (b_j - b_i)(a_i - a_j)^{-1} \mod q $$

- Pollard suggests this random walk: $x_i = t^{a_i}g^{b_i}$

  **Start at arbitrary $a_0, b_0$, $x_0 = t^{a_0}g^{b_0}$:**

  $x_{i+1} = \begin{cases} 
  x_i & 1 \leq x_i < p/3 \\
  x_i^2 & p/3 \leq x_i < 2p/3 \\
  gx_i & 2p/3 \leq x_i < p 
  \end{cases}$

  $(a_{i+1}, b_{i+1}) = \begin{cases} 
  (a_i + 1, b_i) \\
  (2a_i, 2b_i) \\
  (a_i, b_i + 1) 
  \end{cases}$

  Using more intervals works better. (Teske)
Pollard rho analysis

Use Floyd cycle-finding algorithm. Pick an arbitrary starting point and iterate.

- Running time: \( O(\sqrt{q}) \) steps until collision.
- Storage: \( O(1) \).

\[ \begin{align*}
\text{element size} & : O(\log p) \\
\text{ai, bi} & : O(\log q)
\end{align*} \]
More math review: Algebraic rings

- A ring is a set closed under two operations: $+$, $\times$
- $+$ has identity, inverses, associative, commutative
- $\times$ has identity, associative, commutative, might not have inverses
  - **distribution law**

Canonical examples to remember:

- $\mathbb{Z}$
- $\mathbb{Z}_N$, the integers modulo $N$

From last time: $\mathbb{Z}_N^* = \{z \in \mathbb{Z}_N \text{ s.t. } \gcd(z, N) = 1\}$
This is an abelian group under $\times$. 